Journal of Mathematical Imaging and Vision (2020) 62:352–364
https://doi.org/10.1007/s10851-019-00903-1

Deep Neural Networks Motivated by Partial Differential Equations

Lars Ruthotto[1,3] · **Eldad Haber**[2,3]

Received: 30 November 2018 / Accepted: 6 September 2019 / Published online: 18 September 2019

Abstract
Partial differential equations (PDEs) are indispensable for modeling many physical phenomena and also commonly used for solving image processing tasks. In the latter area, PDE-based approaches interpret image data as discretizations of multivariate functions and the output of image processing algorithms as solutions to certain PDEs. Posing image processing problems in the infinite-dimensional setting provides powerful tools for their analysis and solution. For the last few decades, the reinterpretation of classical image processing problems through the PDE lens has been creating multiple celebrated approaches that benefit a vast area of tasks including image segmentation, denoising, registration, and reconstruction. In this paper, we establish a new PDE interpretation of a class of deep convolutional neural networks (CNN) that are commonly used to learn from speech, image, and video data. Our interpretation includes convolution residual neural networks (ResNet), which are among the most promising approaches for tasks such as image classification having improved the state-of-the-art performance in prestigious benchmark challenges. Despite their recent successes, deep ResNets still face some critical challenges associated with their design, immense computational costs and memory requirements, and lack of understanding of their reasoning. Guided by well-established PDE theory, we derive three new ResNet architectures that fall into two new classes: parabolic and hyperbolic CNNs. We demonstrate how PDE theory can provide new insights and algorithms for deep learning and demonstrate the competitiveness of three new CNN architectures using numerical experiments.

Keywords Machine learning · Deep neural networks · Partial differential equations · PDE-constrained optimization · Image classification

1 Introduction

For the last three decades, algorithms inspired by partial differential equations (PDE) have had a profound impact on many processing tasks that involve speech, image, and video data. Adapting PDE models that were traditionally used in physics to perform image processing tasks has led to groundbreaking contributions. An incomplete list of seminal works includes optical flow models for motion estimation [29], nonlinear diffusion models for filtering of images [41], variational methods for image segmentation [1,8,39], and nonlinear edge-preserving denoising [45].

A standard step in PDE-based data processing is interpreting the involved data as discretizations of multivariate functions. Consequently, many operations on the data can be modeled as discretizations of PDE operators acting on the underlying functions. This continuous data model has led to solid mathematical theories for classical data processing tasks obtained by leveraging the rich results from PDEs and variational calculus (e.g., [46]). The continuous perspective has also enabled more abstract formulations that are independent of the actual resolution, which has been exploited to obtain efficient multiscale and multilevel algorithms (e.g., [37]).

In this paper, we establish a new PDE interpretation of deep learning tasks that involve speech, image, and video data as features. Deep learning is a form of machine learning that uses neural networks with many hidden layers [4,34]. Although neural networks date back at least to the 1950s [44], their popularity soared a few years ago when deep neural networks (DNNs) outperformed other machine learning

✉ Lars Ruthotto
lruthotto@emory.edu

Eldad Haber
ehaber@eoas.ubc.ca

[1] Department of Mathematics, Emory University, 400 Dowman Drive, Atlanta, GA 30322, USA

[2] Department of Earth and Ocean Science, The University of British Columbia, Vancouver, BC, Canada

[3] Xtract Technologies Inc., Vancouver, Canada

methods in speech recognition [42] and image classification [28]. Deep learning also led to dramatic improvements in computer vision, e.g., surpassing human performance in image recognition [28,32,34]. These results ignited the recent flare of research in the field. To obtain a PDE interpretation, we use a continuous representation of the images and extend recent works by [21,49], which relate deep learning problems for general data types to ordinary differential equations (ODE).

Deep neural networks filter input features using several layers whose operations consist of element-wise nonlinearities and affine transformations. The main idea of convolutional neural networks (CNN) [33] is to base the affine transformations on convolution operators with compactly supported filters. Supervised learning aims at learning the filters and other parameters, which are also called weights, from training data. CNNs are widely used for solving large-scale learning tasks involving data that represent a discretization of a continuous function, e.g., voice, images, and videos [32,33,35]. By design, each CNN layer exploits the local relation between image information, which simplifies computation [42].

Despite their enormous success, deep CNNs still face critical challenges including designing a CNN architecture that is effective for a practical learning task, which requires many choices. In addition to the number of layers, also called depth of the network, important aspects are the number of convolution filters at each layer, also called the width of the layers, and the connections between those filters. A recent trend is to favor deep over wide networks, aiming at improving generalization, i.e., the performance of the CNN on new examples that were not used during the training [34]. Another key challenge is designing the layer, i.e., choosing the combination of affine transformations and nonlinearities. A practical but costly approach is to consider depth, width, and other properties of the architecture as hyperparameters and jointly infer them with the network weights [26]. Our interpretation of CNN architectures as discretized PDEs provides new mathematical theories to guide the design process. In short, we obtain architectures by discretizing the underlying PDE through adequate time integration methods.

In addition to substantial training costs, deep CNNs face fundamental challenges when it comes to their interpretability and robustness. In particular, CNNs that are used in mission-critical tasks (such as driverless cars) face the challenge of being "explainable." Casting the learning task within nonlinear PDE theory allows us to understand the properties of such networks better. We believe that further research into the mathematical structures presented here will result in a more solid understanding of the networks and will close the gap between deep learning and more mature fields that rely on nonlinear PDEs such as fluid dynamics. A direct impact of our approach can be observed when studying, e.g., adversarial examples. Recent works [40] indicate that the predictions obtained by deep networks can be very sensitive to perturbations of the input images. These findings motivate us to favor networks that are stable, i.e., networks whose output are robust to small perturbations of the input features, similar to what PDE analysis suggests.

In this paper, we consider residual neural networks (ResNet) [24], a very effective type of neural networks. We show that residual CNNs can be interpreted as a discretization of a space-time differential equation. We use this link for analyzing the stability of a network and for motivating new network models that bear similarities with well-known PDEs. Using our framework, we present three new architectures. First, we introduce parabolic CNNs that restrict the forward propagation to dynamics that smooth image features and bear similarities with anisotropic filtering [12,41,48]. Second, we propose hyperbolic CNNs that are inspired by Hamiltonian systems and finally, a third, second-order hyperbolic CNN. As to be expected, those networks have different properties. For example, hyperbolic CNNs approximately preserve the energy in the system, which sets them apart from parabolic networks that smooth the image data, reducing the energy. Computationally, the structure of a hyperbolic forward propagation can be exploited to alleviate the memory burden because hyperbolic dynamics can be made reversible on the continuous and discrete levels. The methods suggested here are closely related to reversible ResNets [9,18].

The remainder of this paper is organized as follows. In Sect. 2, we briefly introduce residual networks and their relation to ordinary and, in the case of convolutional neural networks, partial differential equations. In Sect. 3, we present three novel CNN architectures motivated by PDE theory. Based on our continuous interpretation, we design regularization functionals that enforce the smoothness of the dynamical systems, in Sect. 4. In Sect. 5, we present numerical results for image classification that indicate the competitiveness of our PDE-based architectures. Finally, we highlight some directions for future research in Sect. 6.

2 Residual Networks and Differential Equations

The abstract goal of machine learning is to find a function $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ such that $f(\cdot, \boldsymbol{\theta})$ accurately predicts the result of an observed phenomenon (e.g., the class of an image or a spoken word). The function is parameterized by the weight vector $\boldsymbol{\theta} \in \mathbb{R}^p$ that is trained using examples. In supervised learning, a set of input features $\mathbf{y}_1, \ldots, \mathbf{y}_s \in \mathbb{R}^n$ and output labels $\mathbf{c}_1, \ldots, \mathbf{c}_s \in \mathbb{R}^m$ are available and used to train the model $f(\cdot, \boldsymbol{\theta})$. The output labels are vectors whose components correspond to the estimated probability of a particular example belonging to a given class. As an

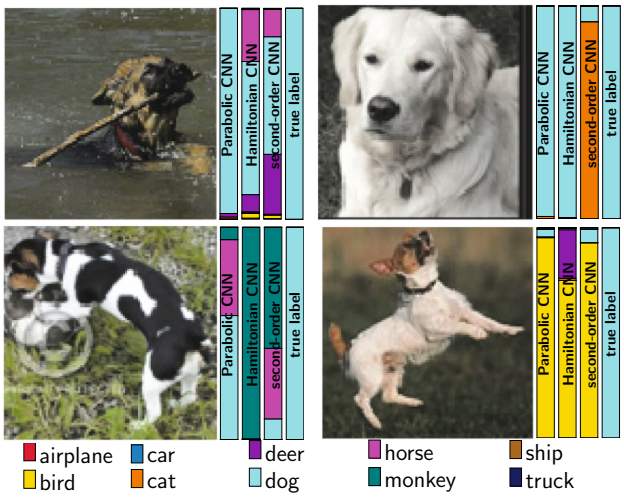

Fig. 1 Classification results of the three proposed CNN architecture for four test images from the STL-10 dataset [14]. The predicted and actual class probabilities are visualized using bar plots on the right of each image. While all networks reach an adequate prediction accuracy between around 79.6% and 80.9% across the whole dataset, predictions for individual images vary in some cases (Color figure online)

example, consider the image classification results in Fig. 1 where the predicted and actual labels are visualized using bar plots. For brevity, we denote the training data by $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_s] \in \mathbb{R}^{n\times s}$ and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_s] \in \mathbb{R}^{m\times s}$.

In deep learning, the function f consists of a concatenation of nonlinear functions called hidden layers. Each layer is composed of affine linear transformations and pointwise nonlinearities and aims at filtering the input features in a way that enables learning. As a fairly general formulation, we consider an extended version of the layer used in [24], which filters the features $\mathbf{Y}$ as follows

$$\mathbf{F}(\boldsymbol{\theta}, \mathbf{Y}) = \mathbf{K}_2(\boldsymbol{\theta}^{(3)})\sigma\left(\mathcal{N}(\mathbf{K}_1(\boldsymbol{\theta}^{(1)})\mathbf{Y}, \boldsymbol{\theta}^{(2)})\right). \tag{1}$$

Here, the parameter vector, $\boldsymbol{\theta}$, is partitioned into three parts where $\boldsymbol{\theta}^{(1)}$ and $\boldsymbol{\theta}^{(3)}$ parameterize the linear operators $\mathbf{K}_1(\cdot) \in \mathbb{R}^{\tilde{w}\times n}$ and $\mathbf{K}_2(\cdot) \in \mathbb{R}^{w_{\text{out}}\times\tilde{w}}$, respectively, and $\boldsymbol{\theta}^{(2)}$ are the parameters of the normalization layer $\mathcal{N}$. The parameters $\tilde{w}$ and w_{out} denote the *width* of the layer, i.e., they correspond to the number of input, intermediate, and output features of this layer. The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied component-wise. Common examples are $\sigma(x) = \tanh(x)$ or the rectified linear unit (ReLU) defined as $\sigma(x) = \max(0, x)$. A deep neural network can be written by concatenating many of the layers given in (1).

When dealing with image data, it is common to group the features into different channels (e.g., for RGB image data there are three channels) and define the operators $\mathbf{K}_1$ and $\mathbf{K}_2$ as block matrices consisting of spatial convolutions. Typically each channel of the output image is computed as a weighted sum of each of the convolved input channels. To give an example, assume that $\mathbf{K}_1$ has three input and two output channels and denote by $\mathbf{K}_1^{(\cdot,\cdot)}(\cdot)$ a standard convolution operator [23]. In this case, we can write $\mathbf{K}_1$ as

$$\mathbf{K}_1(\boldsymbol{\theta}) = \begin{pmatrix} \mathbf{K}_1^{(1,1)}(\boldsymbol{\theta}^{(1,1)}) & \mathbf{K}_1^{(1,2)}(\boldsymbol{\theta}^{(1,2)}) & \mathbf{K}_1^{(1,3)}(\boldsymbol{\theta}^{(1,3)}) \\ \mathbf{K}_1^{(2,1)}(\boldsymbol{\theta}^{(2,1)}) & \mathbf{K}_1^{(2,2)}(\boldsymbol{\theta}^{(2,2)}) & \mathbf{K}_1^{(2,3)}(\boldsymbol{\theta}^{(2,3)}) \end{pmatrix}, \tag{2}$$

where $\boldsymbol{\theta}^{(i,j)}$ denotes the parameters of the stencil of the (i, j)-th convolution operator.

A common choice for $\mathcal{N}$ in (1) is the *batch normalization* layer [30]. This layer computes the empirical mean and standard deviation of each channel in the input images across the spatial dimensions and examples and uses this information to normalize the statistics of the output images. While the coupling of different examples is counter-intuitive, its use is widespread and motivated by empirical evidence showing a faster convergence of training algorithms. The weights $\boldsymbol{\theta}^{(2)}$ represent scaling factors and biases (i.e., constant shifts applied to all pixels in the channel) for each output channel that are applied after the normalization.

ResNets have recently improved the state-of-the-art in several benchmarks including computer vision contests on image classification [28,32,34]. Given the input features $\mathbf{Y}_0 = \mathbf{Y}$, a ResNet block with N layers produces a filtered version $\mathbf{Y}_N$ as follows

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \mathbf{F}(\boldsymbol{\theta}^{(j)}, \mathbf{Y}_j), \quad \text{for} \quad j = 0, 1, \ldots, N-1, \tag{3}$$

where $\boldsymbol{\theta}^{(j)}$ are the weights (convolution stencils and biases) of the jth layer. To emphasize the dependency of this process on the weights, we denote $\mathbf{Y}_N(\boldsymbol{\theta})$.

Note that the dimension of the feature vectors (i.e., the image resolution and the number of channels) is the same across all layers of a ResNets block, which is limiting in many practical applications. Therefore, implementations of deep CNNs contain a concatenation of ResNet blocks with other layers that can change, e.g., the number of channels and the image resolution; see, e.g., [9,24] and Fig. 2.

In image recognition, the goal is to classify the output of (3), $\mathbf{Y}_N(\boldsymbol{\theta})$, using, e.g., a linear classifier modeled by a fully connected layer, i.e., an affine transformation with a dense matrix. To avoid confusion with the ResNet blocks, we denote these transformations as $\mathbf{W}\mathbf{Y}_N(\boldsymbol{\theta}) + (\mathbf{B}_W\boldsymbol{\mu})\mathbf{e}_s^\top$, where the columns of $\mathbf{B}_W$ represent a distributed bias and $\mathbf{e}_s \in \mathbb{R}^s$ is a vector of all ones. The parameters of the network and the classifier are unknown and have to be learned. Thus, the goal of learning is to estimate the network parameters, $\boldsymbol{\theta}$, and the weights of the classifier, $\mathbf{W}$, $\boldsymbol{\mu}$, by approximately solving the optimization problem

$$\min_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \frac{1}{2} S(\mathbf{W}\mathbf{Y}_N(\boldsymbol{\theta}) + (\mathbf{B}_W\boldsymbol{\mu})\mathbf{e}_s^\top, \mathbf{C}) + R(\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}), \tag{4}$$

where S is a loss function, which is convex in its first argument, and R is a convex regularizer discussed below. Typical examples of loss functions are the least-squares function in regression and logistic regression or cross-entropy functions in classification [19].

The optimization problem in (4) is challenging for several reasons. First, it is a high-dimensional and non-convex optimization problem. Therefore one has to be content with local minima. Second, the computational cost per example is high, and the number of examples is large. Third, very deep architectures are prone to problems such as vanishing and exploding gradients [5] that may occur when the discrete forward propagation is unstable [21].

2.1 Residual Networks and ODEs

We derived a continuous interpretation of the filtering provided by ResNets in [21]. Similar observations were made in [10,49]. The ResNet in (3) can be seen as a forward Euler discretization (with a fixed step size of $\delta_t = 1$) of the initial value problem

$$\begin{aligned} \partial_t \mathbf{Y}(\boldsymbol{\theta}, t) &= \mathbf{F}(\boldsymbol{\theta}(t), \mathbf{Y}(t)), \quad \text{for } t \in (0, T] \\ \mathbf{Y}(\boldsymbol{\theta}, 0) &= \mathbf{Y}_0. \end{aligned} \tag{5}$$

Here, we introduce an artificial time $t \in [0, T]$. The depth of the network is related to the arbitrary final time T and the magnitude of the matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ in (1). This observation shows the relation between the learning problem (4) and parameter estimation of a system of nonlinear ordinary differential equations. Note that this interpretation does not assume any particular structure of the layer $\mathbf{F}$.

The continuous interpretation of ResNets can be exploited in several ways. One idea is to accelerate training by solving a hierarchy of optimization problems that gradually introduce new time discretization points for the weights, $\boldsymbol{\theta}$ [22]. Also, new numerical solvers based on optimal control theory have been proposed in [36]. Another recent work [11] presents an optimize-discretize approach with sophisticated time integrators to solve the forward propagation and the adjoint problem (in this context commonly called back-propagation), which is needed to compute derivatives of the objective function with respect to the network weights.

2.2 Convolutional ResNets and PDEs

In the following, we consider learning tasks involving features given by speech, image, or video data. For these problems, the input features, $\mathbf{Y}$, can be seen as a discretization of a continuous function $Y(x)$. We assume that the matrices $\mathbf{K}_1 \in \mathbb{R}^{\tilde{w} \times w_{\text{in}}}$ and $\mathbf{K}_2 \in \mathbb{R}^{w_{\text{out}} \times \tilde{w}}$ in (1) represent convolution operators [23].

We now show that a particular class of deep residual CNNs can be interpreted as nonlinear systems of PDEs. For ease of notation, we first consider a one-dimensional convolution of a feature with one channel and then outline how the result extends to higher space dimensions and multiple channels.

Assume that the vector $\mathbf{y} \in \mathbb{R}^n$ represents a one-dimensional grid function obtained by discretizing $y : [0, 1] \to \mathbb{R}$ at the cell-centers of a regular grid with n cells and a mesh size $h = 1/n$, i.e., for $i = 1, 2, \ldots, n$

$$\mathbf{y} = [y(x_1), \ldots, y(x_n)]^\top \quad \text{with} \quad x_i = \left(i - \frac{1}{2}\right) h.$$

Assume, e.g., that the operator $\mathbf{K}_1 = \mathbf{K}_1(\boldsymbol{\theta}) \in \mathbb{R}^{n \times n}$ in (1) is parameterized by the stencil $\boldsymbol{\theta} \in \mathbb{R}^3$. Applying a coordinate change, we see that

$$\begin{aligned} \mathbf{K}_1(\boldsymbol{\theta})\mathbf{y} &= [\boldsymbol{\theta}_1\, \boldsymbol{\theta}_2\, \boldsymbol{\theta}_3] * \mathbf{y} \\ &= \left(\frac{\boldsymbol{\beta}_1}{4}[1\,2\,1] + \frac{\boldsymbol{\beta}_2}{2h}[-1\,0\,1] + \frac{\boldsymbol{\beta}_3}{h^2}[-1\,2\,-1] \right) * \mathbf{y}. \end{aligned}$$

Here, the weights $\boldsymbol{\beta} \in \mathbb{R}^3$ are given by

$$\begin{pmatrix} \frac{1}{4} & -\frac{1}{2h} & -\frac{1}{h^2} \\ \frac{1}{2} & 0 & \frac{2}{h^2} \\ \frac{1}{4} & \frac{1}{2h} & -\frac{1}{h^2} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \boldsymbol{\beta}_3 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \boldsymbol{\theta}_3 \end{pmatrix},$$

which is a non-singular linear system for any $h > 0$. We denote by $\boldsymbol{\beta}(\boldsymbol{\theta})$ the unique solution of this linear system. Upon taking the limit, $h \to 0$, this observation motivates one to parameterize the convolution operator as

$$\mathbf{K}_1(\boldsymbol{\theta}) = \boldsymbol{\beta}_1(\boldsymbol{\theta}) + \boldsymbol{\beta}_2(\boldsymbol{\theta})\partial_x + \boldsymbol{\beta}_3(\boldsymbol{\theta})\partial_x^2.$$

The individual terms in the transformation matrix correspond to reaction, convection, diffusion and the bias term in (1) is a source/sink term, respectively. Note that higher-order derivatives can be generated by multiplying different convolution operators or increasing the stencil size.

This simple observation exposes the dependence of learned weights on the image resolution, which can be exploited in practice, e.g., by multiscale training strategies [22]. Here, the idea is to train a sequence of networks using a coarse-to-fine hierarchy of image resolutions (often called *image pyramid*). Since both the number of operations and the memory required in training is proportional to the image size, this leads to immediate savings during training but also allows one to coarsen already trained networks to enable efficient evaluation. In addition to computational benefits, ignoring fine-scale features when training on the coarse grid can also reduce the risk of being trapped in an undesirable local minimum, which is an observation also made in other image processing applications.

Our argument extends to higher spatial dimensions. In 2D, e.g., we can relate the 3×3 stencil parametrized by $\boldsymbol{\theta} \in \mathbb{R}^9$ to

$$\begin{aligned}\mathbf{K}_1(\boldsymbol{\theta}) = \boldsymbol{\beta}_1(\boldsymbol{\theta}) + \boldsymbol{\beta}_2(\boldsymbol{\theta})\partial_x + \boldsymbol{\beta}_3(\boldsymbol{\theta})\partial_y \\ + \boldsymbol{\beta}_4(\boldsymbol{\theta})\partial_x^2 + \boldsymbol{\beta}_5(\boldsymbol{\theta})\partial_y^2 + \boldsymbol{\beta}_6(\boldsymbol{\theta})\partial_x\partial_y \\ + \boldsymbol{\beta}_7(\boldsymbol{\theta})\partial_x^2\partial_y + \boldsymbol{\beta}_8(\boldsymbol{\theta})\partial_x\partial_y^2 + \boldsymbol{\beta}_9(\boldsymbol{\theta})\partial_x^2\partial_y^2.\end{aligned}$$

To obtain a fully continuous model for the layer in (1), we proceed the same way with $\mathbf{K}_2$. In view of (2), we note that when the number of input and output channels is larger than one, $\mathbf{K}_1$ and $\mathbf{K}_2$ lead to a system of coupled partial differential operators.

Given the continuous space-time interpretation of CNN, we view the optimization problem (4) as an optimal control problem and, similarly, see learning as a parameter estimation problem for the time-dependent nonlinear PDE (5). Developing efficient numerical methods for solving PDE-constrained optimization problems arising in optimal control and parameter estimation has been a fruitful research endeavor and led to many advances in science and engineering; for recent overviews see, e.g., [6,7,27]. Using the theoretical and algorithmic framework of optimal control in machine learning applications has gained some traction only recently; see, e.g., [9,11,21,36,49].

3 Deep Neural Networks Motivated by PDEs

It is well-known that not every time-dependent PDE is stable with respect to perturbations of the initial conditions [2]. Here, we say that the forward propagation in (5) is stable if there is a constant $M > 0$ *independent of* T such that

$$\|\mathbf{Y}(\boldsymbol{\theta}, T) - \tilde{\mathbf{Y}}(\boldsymbol{\theta}, T)\|_F \leq M\|\mathbf{Y}_0 - \tilde{\mathbf{Y}}_0\|_F, \tag{6}$$

where $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ are the solutions of (5) for the initial values $\mathbf{Y}_0$, $\tilde{\mathbf{Y}}_0$, respectively, and $\|\cdot\|_F$ is the Frobenius norm. The stability of the forward propagation depends on the values of the weights $\boldsymbol{\theta}$ that are chosen by solving (4). In the context of learning, the stability of the network is critical to provide robustness to small perturbations of the input images. In addition to image noise, perturbations could also be added deliberately to mislead the network's prediction by an adversary. There is some recent evidence showing the existence of such perturbations that reliably mislead deep networks by being barely noticeable to a human observer; see, e.g., [20,38,40]. Networks that satisfy (6) are Lipschitz continuous with a Lipschitz constant that is independent of T. Lipschitz continuity has been recently shown to be important for generalization and robustness of deep neural networks; see, e.g., [13,15,16] and references therein.

To ensure the stability of the network for all possible weights, we propose to restrict the space of CNNs. As examples of this general idea, we present three new types of residual CNNs that are motivated by parabolic and first- and second-order hyperbolic PDEs, respectively. The construction of our networks guarantees that under some assumptions the networks are stable forward and, for the hyperbolic network, stable backward in time.

Though it is common practice to model $\mathbf{K}_1$ and $\mathbf{K}_2$ in (1) independently, we note that it is, in general, hard to show the stability of the resulting network. This is because, the Jacobian of $\mathbf{F}(\boldsymbol{\theta}, \mathbf{Y})$ with respect to the features has the form

$$\mathbf{J}_\mathbf{Y}\mathbf{F} = \mathbf{K}_2(\boldsymbol{\theta})\ \mathrm{diag}(\sigma'(\mathbf{K}_1(\boldsymbol{\theta})\mathbf{Y}))\ \mathbf{K}_1(\boldsymbol{\theta}),$$

where σ' denotes the derivatives of the pointwise nonlinearity and for simplicity we assume $\mathcal{N}(\mathbf{Y}) = \mathbf{Y}$. Even in this simplified setting, the spectral properties of $\mathbf{J}_\mathbf{Y}$, which impact the stability, are unknown for arbitrary choices of $\mathbf{K}_1$ and $\mathbf{K}_2$.

As one way to obtain a stable network, we introduce a symmetric version of the layer in (1) by choosing $\mathbf{K}_2 = -\mathbf{K}_1^\top$ in (1). To simplify our notation, we drop the subscript of the operator and define the symmetric layer

$$\mathbf{F}_{\mathrm{sym}}(\boldsymbol{\theta}, \mathbf{Y}) = -\mathbf{K}(\boldsymbol{\theta})^\top \sigma\left(\mathcal{N}(\mathbf{K}(\boldsymbol{\theta})\mathbf{Y}, \boldsymbol{\theta})\right). \tag{7}$$

It is straightforward to verify that this choice leads to a negative semi-definite Jacobian for any non-decreasing activation function. As we see next, this choice also allows us to link the discrete network to different types of PDEs.

3.1 Parabolic CNN

We define the parabolic CNN by using the symmetric layer from (7) in the forward propagation, i.e., in the standard ResNet we replace the dynamic in (5) by

$$\partial_t \mathbf{Y}(\boldsymbol{\theta}, t) = \mathbf{F}_{\mathrm{sym}}(\boldsymbol{\theta}(t), \mathbf{Y}(t)), \quad \text{for } t \in (0, T]. \tag{8}$$

Note that (8) is equivalent to the heat equation if $\sigma(x) = x$, $\mathcal{N}(\mathbf{Y}) = \mathbf{Y}$ and $\mathbf{K}(t) = \nabla$. This motivates us to refer to this network as a parabolic CNN. This is not the only possible interpretation, e.g., for other cases in which $\mathbf{K}$ is constant in time (8) can be seen as a gradient flow. Nonlinear parabolic PDEs are widely used, e.g., to filter images [12,41,48] and our interpretation implies that the networks can be viewed as an extension of such methods.

The similarity to the heat equation motivates us to introduce a new normalization layer motivated by total variation denoising. For a single example $\mathbf{y} \in \mathbb{R}^n$ that can be grouped into c channels, we define

$$\mathcal{N}_{\text{tv}}(\mathbf{y}) = \operatorname{diag}\left(\frac{1}{\mathbf{A}^\top\sqrt{(\mathbf{A}\mathbf{y})^2+\epsilon}}\right)\mathbf{y}, \tag{9}$$

where the operator $\mathbf{A} \in \mathbb{R}^{\frac{n}{c}\times n}$ computes the sum over all c channels for each pixel, the square, square root, and the division are defined component-wise, and the constant $0 < \epsilon \ll 1$ is fixed. As for the batch norm layer, we implement $\mathcal{N}_{\text{tv}}$ with trainable weights corresponding to global scaling factors and biases for each channel. In the case that the convolution is reduced to a discrete gradient, $\mathcal{N}_{\text{tv}}$ leads to the regular dynamics in TV denoising.

Stability Parabolic PDEs have a well-known decay property that renders them robust to perturbations of the initial conditions. For the parabolic CNN in (8), we can show the following stability result.

Theorem 1 *If the activation function σ is monotonically non-decreasing, then the forward propagation through a parabolic CNN satisfies* (6).

Proof For ease of notation, we assume that no normalization layer is used, i.e., $\mathcal{N}(\mathbf{Y}) = \mathbf{Y}$ in (8). We then show that $\mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t), \mathbf{Y})$ is a monotone operator. Let $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ be the solutions of (5) for the initial values $\mathbf{Y}_0$, $\tilde{\mathbf{Y}}_0$, respectively.

Note that for all $t \in [0, T]$

$$-\left(\sigma(\mathbf{K}(t)\mathbf{Y}) - \sigma(\mathbf{K}(t)\tilde{\mathbf{Y}}), \mathbf{K}(t)(\mathbf{Y}-\tilde{\mathbf{Y}})\right) \leq 0.$$

Where $(\cdot,\cdot)$ is the standard inner product and the inequality follows from the monotonicity of the activation function, which shows that

$$\partial_t \|\mathbf{Y}(t) - \tilde{\mathbf{Y}}(t)\|_F^2 \leq 0.$$

Integrating this inequality over $[0, T]$ yields stability as in (6). The proof extends straightforwardly to cases when a normalization layer with scaling and bias is included. □

One way to discretize the parabolic forward propagation (8) is using the forward Euler method. Denoting the time step size by $\delta_t > 0$ this reads

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t_j), \mathbf{Y}_j), \quad j = 0, 1, \ldots, N-1, \tag{10}$$

where $t_j = j\delta_t$. For a linear dynamic $\partial_t \mathbf{y}(t) = \mathbf{J}\mathbf{y}(t)$, the forward Euler method is stable if δ_t satisfies

$$\max_{i=1,2,\ldots,n} |1 + \delta_t \lambda_i(\mathbf{J})| \leq 1$$

and accurate if δ_t is chosen small enough to capture the dynamics of the system. Here, $\lambda_i(\mathbf{J})$ denotes the ith eigenvalue of $\mathbf{J}$; see, e.g., [2,3]. A rigorous stability analysis of the discrete forward propagation in (10) is more complicated since the underlying PDE is nonlinear and time-dependent. However, the result above provides some intuition. Here, we can linearize the forward propagation using the Jacobian $\mathbf{J}(t_j) = (\nabla_{\mathbf{y}}\mathbf{F}_{\text{sym}})^\top$ at the jth time point t_j. Assuming, for simplicity, that no normalization layer is used, this reads

$$\mathbf{J}(t_j) = -\mathbf{K}^\top(\boldsymbol{\theta}^{(1)}(t_j))\,\mathbf{D}(t_j)\mathbf{K}(\boldsymbol{\theta}^{(1)}(t_j)),$$
$$\text{with} \quad \mathbf{D}(t) = \operatorname{diag}\left(\sigma'\left(\mathbf{K}(\boldsymbol{\theta}^{(1)}(t))\mathbf{y}(t)\right)\right).$$

If the activation function is monotonically non-decreasing, then $\sigma'(\cdot) \geq 0$ everywhere. In this case, all eigenvalues of $\mathbf{J}(t_j)$ are real and bounded above by zero since $\mathbf{J}(t_j)$ is also symmetric. Thus, there is an appropriate δ_t that renders the linearized discrete forward propagation stable. To achieve stability of the linear problem, we limit the magnitude of elements in $\mathbf{K}$ by using Tikhonov regularization and imposing bound constraints to the optimization problem (4). Also, to minimize the difference between the linearized forward propagation and the actual dynamics, we penalize large temporal changes through regularization; see Sect. 4.

3.2 Hyperbolic CNNs

Different types of networks can be obtained by considering hyperbolic PDEs. In this section, we present two CNN architectures that are inspired by hyperbolic systems. A favorable feature of hyperbolic equations is their reversibility. Reversibility allows us to avoid storage of intermediate network states, thus achieving higher memory efficiency. Reversibility is particularly important for very deep networks where memory limitation can hinder training; see [18] and [9].

Hamiltonian CNNs Introducing an auxiliary variable $\mathbf{Z}$, we consider the dynamics

$$\partial_t \mathbf{Y}(t) = \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t), \mathbf{Z}(t)), \qquad \mathbf{Y}(0) = \mathbf{Y}_0$$
$$\partial_t \mathbf{Z}(t) = -\mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t), \mathbf{Y}(t)), \qquad \mathbf{Z}(0) = \mathbf{Z}_0.$$

The dimensions and the values of $\mathbf{Z}_0$ can be chosen in different ways, e.g., by partitioning the channels of the original features into $\mathbf{Y}_0$ and $\mathbf{Z}_0$. This approach is used in our numerical experiments in which we split the number of channels into two. We showed in [9] that the eigenvalues of the associated Jacobian are imaginary. When assuming that $\boldsymbol{\theta}^{(1)}$ and $\boldsymbol{\theta}^{(2)}$ are constant in time, stability as defined in (6) is obtained. A more precise stability result can be established by analyzing the kinematic eigenvalues of the forward propagation [3].

We discretize the dynamic using the symplectic Verlet integration (see, e.g., [2] for details)

$$\begin{aligned}\mathbf{Y}_{j+1} &= \mathbf{Y}_j + \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t_j), \mathbf{Z}_j),\\ \mathbf{Z}_{j+1} &= \mathbf{Z}_j - \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t_j), \mathbf{Y}_{j+1}),\end{aligned} \tag{11}$$

for $j = 0, 1, \ldots, N-1$ using a fixed step size $\delta_t > 0$. This dynamic is *reversible*, i.e., given $\mathbf{Y}_N, \mathbf{Y}_{N-1}$ and $\mathbf{Z}_N, \mathbf{Z}_{N-1}$ it can also be computed backwards

$$\begin{aligned}\mathbf{Z}_j &= \mathbf{Z}_{j+1} + \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(2)}(t_j), \mathbf{Y}_{j+1})\\ \mathbf{Y}_j &= \mathbf{Y}_{j+1} - \delta_t \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}^{(1)}(t_j), \mathbf{Z}_j),\end{aligned}$$

for $j = N-1, N-2, \ldots, 0$. These operations are numerically stable for the Hamiltonian CNN; see [9] for details.

Second-order CNNs An alternative way to obtain hyperbolic CNNs is by using a second-order dynamics

$$\begin{aligned}\partial_t^2 \mathbf{Y}(t) &= \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t), \mathbf{Y}(t)),\\ \mathbf{Y}(0) &= \mathbf{Y}_0, \;\; \partial_t \mathbf{Y}(0) = 0.\end{aligned} \tag{12}$$

The resulting forward propagation is associated with a nonlinear version of the telegraph equation [43], which describes the propagation of signals through networks. Hence, one could claim that second-order networks better mimic biological networks and are therefore more appropriate than first-order networks for approaches that aim at imitating the propagation through biological networks.

We discretize the second-order network using the Leapfrog method. For $j = 0, 1, \ldots, N-1$ and $\delta_t > 0$ fixed this reads

$$\mathbf{Y}_{j+1} = 2\mathbf{Y}_j - \mathbf{Y}_{j-1} + \delta_t^2 \mathbf{F}_{\text{sym}}(\boldsymbol{\theta}(t_j), \mathbf{Y}_j).$$

We set $\mathbf{Y}_{-1} = \mathbf{Y}_0$ to denote the initial condition. As the symplectic integration in (11), this scheme is reversible under similar conditions.

We show that the second-order network is stable in the sense of (6) when we assume stationary weights. In our experiments, we use regularization to limit the magnitude of $\partial_t \boldsymbol{\theta}(t)$ and ideally obtain piecewise constant weights. A rigorous analysis for the general case is more complicated and an item of future work.

Theorem 2 *Let $\boldsymbol{\theta}(t)$ be constant in time and assume that the activation function satisfies $|\sigma(x)| \leq |x|$ for all x. Then, the forward propagation through the second-order network satisfies* (6).

Proof For brevity, we denote $\mathbf{K} = \mathbf{K}(\boldsymbol{\theta}(t))$ and consider the forward propagation of a single example. Let $\mathbf{y} : [0, T] \to \mathbb{R}^n$ be a solution to (12) and consider the energy

$$\mathcal{E}(t) = \frac{1}{2}\left((\partial_t \mathbf{y}(t))^\top \partial_t \mathbf{y}(t) + (\mathbf{K}\mathbf{y}(t))^\top \sigma(\mathbf{K}\mathbf{y}(t))\right). \tag{13}$$

Given that $|\sigma(x)| \leq |x|$ for all x by assumption, this energy can be bounded as follows

$$\begin{aligned}\mathcal{E}(t) &\leq \mathcal{E}_{\text{lin}}(t)\\ &= \frac{1}{2}\left((\partial_t \mathbf{u}(t))^\top \partial_t \mathbf{u}(t) + (\mathbf{K}\mathbf{u}(t))^\top (\mathbf{K}\mathbf{u}(t))\right),\end{aligned}$$

where $\mathcal{E}_{\text{lin}}$ is the energy associated with the linear wave-like hyperbolic equation

$$\partial_t^2 \mathbf{u}(t) = -\mathbf{K}^\top \mathbf{K}\mathbf{u}(t), \;\; \mathbf{u}(0) = \mathbf{y}_0, \;\; \partial_t \mathbf{u}(0) = 0.$$

Since by assumption $\mathbf{K}$ is constant in time, we have that

$$\partial_t \mathcal{E}_{\text{lin}}(t) = \partial_t \mathbf{u}(t)^\top \left(\partial_t^2 \mathbf{u}(t) + \mathbf{K}^\top \mathbf{K}\mathbf{u}(t)\right) = 0.$$

Thus, the energy of the hyperbolic network in (13) is positive and bounded from above by the energy of the linear wave equation. Applying this argument to the initial condition $\mathbf{y}_0 - \tilde{\mathbf{y}}_0$, we derive (6) and thus the forward propagation is stable.

4 Regularization

The proposed continuous interpretation of the CNNs also provides new perspectives on regularization. To enforce stability of the forward propagation, the linear operator $\mathbf{K}$ in (7) should not change drastically in time. This suggests adding a smoothness regularizer in time. In [21], a H^1-seminorm was used to smooth kernels over time to avoid overfitting. A theoretically more appropriate function space consists of all kernels that are piecewise constant in time. To this end, we introduce the regularizer

$$\begin{aligned}R(\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}) = \alpha_1 &\int_0^T \phi_\tau(\partial_t \boldsymbol{\theta}(t)) dt\\ &+ \frac{\alpha_2}{2}\left(\int_0^T \|\boldsymbol{\theta}(t)\|^2 dt + \|\mathbf{W}\|_F^2 + \|\boldsymbol{\mu}\|^2\right),\end{aligned} \tag{14}$$

where the function $\phi_\tau(x) = \sqrt{x^2 + \tau}$ is a smoothed ℓ_1-norm with conditioning parameter $\tau > 0$. The first term of R can be seen as a total variation [45] penalty in time that favors piecewise constant dynamics. Here, $\alpha_1, \alpha_2 \geq 0$ are regularization parameters that are assumed to be fixed.

A second important aspect of stability is to keep the time step sufficiently small. Since δ_t can be absorbed in $\mathbf{K}$ we use the box constraint $-1 \leq \boldsymbol{\theta}^{(1)}(t_j) \leq 1$ for all j, and fix the time step size to $\delta_t = 1$ in our numerical experiments.

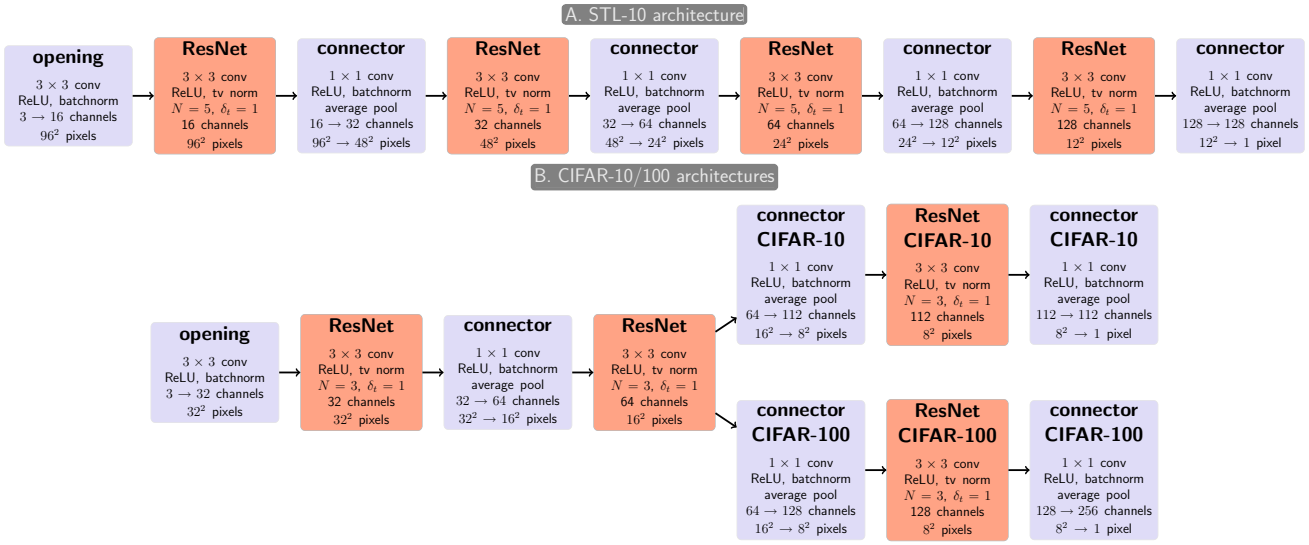

Fig. 2 Overview of network architectures for the STL-10 (top row) and CIFAR-10/100 (bottom row) image classification problems. The architectures consist of ResNet blocks (red) that represent the parabolic, Hamiltonian, and second-order dynamics, respectively. The networks also contain an opening layer and several connector layers (blue) that increase the number of channels and reduce the image resolution (Color figure online)

5 Numerical Experiments

We demonstrate the potential of the proposed architectures using the common image classification benchmarks STL-10 [14], CIFAR-10, and CIFAR-100 [31]. Instead of beating state-of-the-art results on these competitive datasets, our central goal is to show that, despite their modeling restrictions, our new network types achieve competitive results. We use our basic architecture for all experiments, do not excessively tune hyperparameters individually for each case, and employ a simple data augmentation technique consisting of random flipping and cropping.

Network Architecture Our architecture is similar to the ones in [9,24] and contains an opening layer, followed by several blocks each containing a few time steps of a ResNet and a connector that increases the width of the CNN and coarsens the images. Our focus is on the different options for defining the ResNet block using parabolic and hyperbolic networks. To this end, we choose the same basic components for the opening and connecting layers. The opening layer increases the number of channels from three (for RGB image data) to the number of channels of the first ResNet using convolution operators with 3×3 stencils, a batch normalization layer and a ReLU activation function. We build the connecting layers using 1×1 convolution operators that increase the number of channels, a batch normalization layer, a ReLU activation, and an average pooling operator that coarsens the images by a factor of two. Finally, we obtain the output features $\mathbf{Y}(\boldsymbol{\theta})$ by averaging the features of each channel to ensure translation-invariance. The ResNet blocks use the symmetric layer (7) including the total variation normalization (9) with $\epsilon = 10^{-3}$. The network architectures are illustrated in Fig. 2. The classifier is modeled using a fully connected layer, a softmax transformation, and a cross-entropy loss.

Training Algorithm In order to estimate the weights, we use a standard stochastic gradient descent (SGD) method with a momentum of 0.9. We use a piecewise constant step size (in this context also called learning rate). We choose an initial learning rate of 0.05 and divided it by a factor of 1.5 at a priori specified epochs, after every 10th epoch for CIFAR-10 and STL-10 and after every 20th epoch for CIFAR-100. The training is stopped when the training loss drops below 0.01 (indicating overfitting), or a maximum number of epochs is reached (180 for STL-10 and CIFAR-10 and 340 for CIFAR-100). In all examples, the SGD steps are computed using mini-batches consisting of 32 randomly chosen examples.

For data augmentation, we apply a random horizontal flip (50% probability), pad the images by a factor of 1/16 with zeros into all directions and randomly crop the image by 1/8 of the pixels, counting from the lower-left corner. The training is performed using the open-source software Meganet on a workstation running Ubuntu 16.04 and MATLAB 2018b with two Intel(R) Xeon(R) CPU E5-2620 v4 and 64 GB of RAM. We use a NVIDIA Titan X GPU for accelerating the computation through the frameworks CUDA 9.1 and CuDNN 7.0. This package is designed as an academic and teaching tool and not optimized for efficiency. Running up to three instances on the same device, the average time to complete an epoch is about 1.5 minutes for STL-10 and 5 minutes for the CIFAR examples.

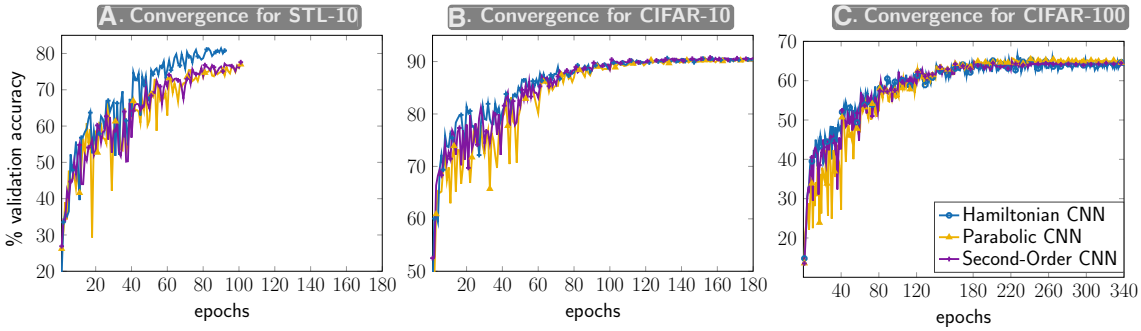

Fig. 3 Performance of the training algorithm for the three proposed architectures applied to the STL-10 (left), CIFAR-10 (middle) and CIFAR-100 (right) datasets. We use randomly choose 80% of the training images to update the weights using SGD. The plots show the validation accuracy computed using the remaining images after every epoch. In these examples, we did not observe considerable overfitting and note that the weights from the final epoch led to adequate validation accuracies (Color figure online)

We demonstrate the effectiveness of this strategy using a simple cross-validation approach that updates the weights using 80% of the training data and monitors the loss computed using 20% of the training data until no significant overfitting is observed; we plot the performance of the networks on the validation data after each epoch in Fig. 3. In the STL-10 experiment (left subplot), the iterations were stopped after around 100 epochs since the training loss was low, whereas for the CIFAR-10/100 experiments (middle, right subplots) the maximum number of epochs was reached. In all cases, the weights at the final epoch are nearly optimal with respect to the validation datasets, which indicates that the learning rate schedule and stopping criteria are adequate to prevent overfitting.

Results for STL-10 The STL-10 dataset [14] contains 13,000 digital color images of size 96×96 that are evenly divided into ten categories, which can be inferred from Fig. 1. The dataset is split into 5,000 training and 8,000 test images. The STL-10 data is a popular benchmark test for image classification algorithms and challenging due to the relatively small number of training images.

For each dynamic, the network uses four ResNet blocks with 16, 32, 64, and 128 channels and image sizes of 96×96, 48×48, 24×24, 12×12, respectively. Within each ResNet block, we perform five time steps with a step size of $\delta_t = 1$ and include a total variation normalization layer and ReLU activation. This architecture leads to 521,594 trainable weights for the Hamiltonian network and 1,011,194 weights for the parabolic and second-order network, respectively. The reversibility of the Hamiltonian and second-order network can be used to reduce the memory consumption of the ResNet blocks during training, e.g., by re-computing the features and activations at the second, third, and fourth layer. This would result in an approximately 60% reduction of memory required in those blocks. Additional savings can be realized by avoiding duplicate storage with connecting layers. As these savings become more drastic and important for very deep architectures (see [9]), we do not exploit the reversibility in our experiments.

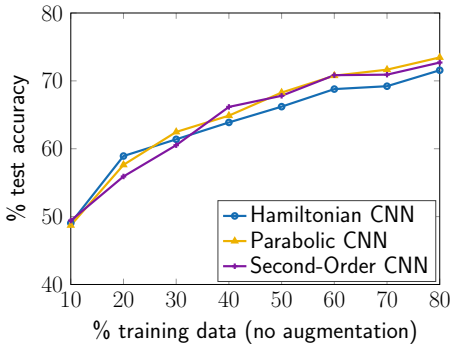

Fig. 4 Improvement of the test accuracy when increasing the number of training images in the STL-10 dataset (from 10% to 80% in increments of 10%). We first train the weights of the three proposed architectures and plot the test accuracy of the final iterate. Here, we do not use any data augmentation. Expectedly, the generalization improves as more images are used for all architectures (Color figure online)

We note that our networks are smaller than commonly used ResNets, e.g., the architectures in [9] contain about two million parameters. Reducing the number of parameters is essential during training and, e.g., when trained networks have to be deployed on devices with limited memory. Another difference to networks in this work is the use of the total variation normalization instead of the batch normalization in the ResNet layers. This removes the coupling between different examples in a batch that is introduced by the batch norm and increases the potential for parallelization. The regularization parameters are $\alpha_1 = 4 \cdot 10^{-4}$ and $\alpha_2 = 1 \cdot 10^{-4}$.

To show how the generalization improves as more training data becomes available, we train the network with an increasing number of examples that we choose randomly from the training dataset. We also randomly sample 1000 examples from the remaining training data to build a validation set, which we use to monitor the performance after each full

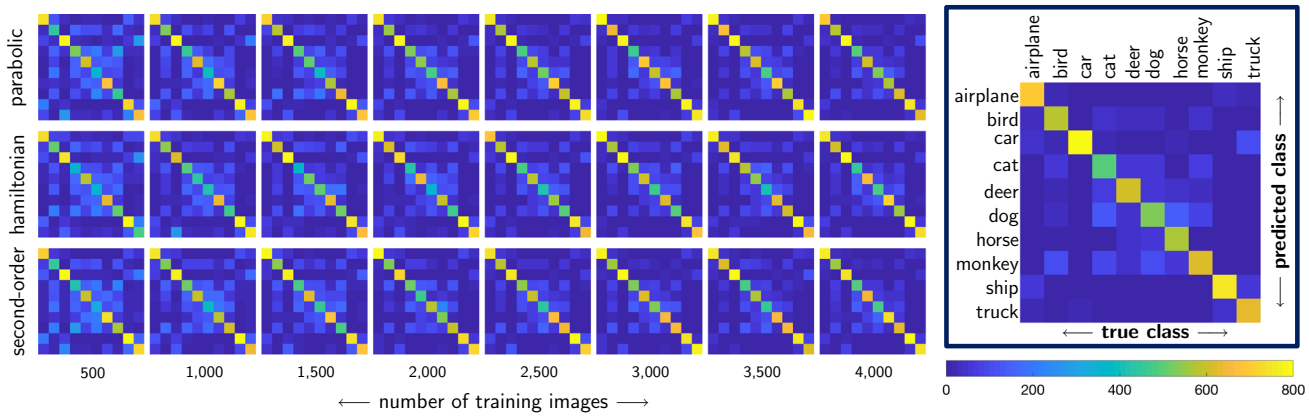

Fig. 5 Confusion matrices for classifiers obtained using the three proposed architectures (row-wise) for an increasing number of training data from the STL-10 dataset (column-wise). The (i, j)th element of the 10×10 confusion matrix counts the number of images of class i for which the predicted class is j. We use the entire test data set, which contains 800 images per class (Color figure online)

epoch. We use no data augmentation in this experiment. In all cases, the training accuracy was close to 100%. After the training, we compute the accuracy of the networks parameterized by the weights that performed best on the validation data for all the 8000 test images; see Fig. 4. The predictions of the three networks may vary for single examples without any apparent pattern; see also Fig. 1. However, overall their performance and convergence are comparable which leads to similarities in the confusion matrices; see Fig. 5.

To show the overall performance of the networks, we train the networks using all 5000 training images and no cross-validation. For data augmentation, we use horizontal flipping and random cropping. Our training strategy found weights that almost perfectly fit the training data (defined by a loss of less than 0.01) after 97, 91, and 106 epoch for the parabolic, Hamiltonian, and second-order CNN, respectively. After the training, we compute the loss and classification accuracy for all 8000 test images. For this example, the parabolic and Hamiltonian network perform slightly superior to the second-order network 80.9% and 80.4% versus 79.6% test classification accuracy, respectively. It is important to emphasize that the Hamiltonian network uses only about half as many trainable weights as the other two networks. These results are competitive with the results reported, e.g., in [17,50] and slightly inferior of the ones we achieved with a larger architecture and the use of batch norm in [9]. Our results could possibly be further improved by fine-tuning of hyperparameters such as step size, number of time steps, and width of the network may achieve additional improvements for each dynamic.

Results for CIFAR-10/100 For additional comparison of the proposed architectures, we use the CIFAR-10 and CIFAR-100 datasets [31]. Each of these datasets consists of 60,000 labeled RGB images of size 32×32 that are chosen from the 80 million tiny images dataset [47]. In both cases, we use 50,000 images for training and validation and keep the remaining 10,000 to test the generalization of the trained weights. While CIFAR-10 consists of ten categories, CIFAR-100 contains 100 categories, which renders the classification problem more challenging.

Our architectures contain three blocks of parabolic or hyperbolic networks between which the image size is reduced from 32×32 to 8×8. For the simpler CIFAR-10 problem, we use a narrower network with 32, 64, 112 channels while for CIFAR-100 we use more channels (32, 64, and 128) and add a final connecting layer that increases the number of channels to 256. This leads to networks whose number of trainable weights vary between 264,106 and 502,570; see also Table 1. As regularization parameters, we use $\alpha_1 = 2 \cdot 10^{-4}$ and $\alpha_2 = 2 \cdot 10^{-4}$, which is similar to [9].

As for the STL-10 data set, the three proposed architectures achieved comparable results on these benchmarks; see convergence plots in Fig. 3 and test accuracies in Table 1. In our experiments, the optimization is stopped after the maximum number of epochs (180 for CIFAR-10 and 340 for CIFAR-100). Additional tuning of the learning rate, regularization parameter, and other hyperparameters may further improve the results shown here. Using similar networks and the less interpretable batch normalization in the ResNet blocks, we achieved about 5% higher accuracy on CIFAR-10 and 9% higher accuracy on CIFAR-100 in [9].

6 Discussion and Outlook

In this paper, we establish a link between deep residual convolutional neural networks and PDEs. The relation provides a general framework for designing, analyzing, and training those CNNs. It also exposes the dependence of learned weights on the image resolution used in training.

Table 1 Summary of numerical results for the STL-10, CIFAR-10, and CIFAR-100 datasets

	STL-10		CIFAR-10		CIFAR-100	
	Number of weights (M)	Test data (8000) accuracy %(loss)	Number of weights (M)	Test data (10,000) accuracy %(loss)	Number of weights (M)	Test data (10,000) accuracy %(loss)
Parabolic	1.01	80.9% (0.726)	0.50	90.5% (0.316)	0.65	67.4% (1.185)
Hamiltonian	0.52	80.4% (0.770)	0.26	90.7% (0.334)	0.36	67.1% (1.208)
Second-order	1.01	79.6% (0.770)	0.50	90.6% (0.329)	0.65	66.9% (1.281)
Hamiltonian [9]	1.28	85.5% (n/a)	0.43	92.8% (n/a)	0.44	71.0% (n/a)
Leapfrog [9]	2.44	84.6% (n/a)	0.50	91.9% (n/a)	0.51	69.1% (n/a)
ResNet-110 [25]			1.7	93.4% (n/a)	1.7	74.8% (n/a)
Wide ResNet [51]			0.6	93.2% (n/a)	0.6	69.1% (n/a)

Using the hyperparameters chosen by cross-validation, we train the networks on the entire training data. After training, we compute and report the classification accuracy and the value of cross-entropy loss (in brackets where reported) for the test data. To this end, we use the weights from the final epoch of SGD. We also report the number of trainable weights for each network and for comparison state results from the literature achieved with similarly sized or larger architectures

Exemplarily, we derive three PDE-based network architectures that are forward stable (the parabolic network) and forward-backward stable (the hyperbolic networks) under some assumptions.

It is well known that different types of PDEs have different properties. For example, linear parabolic PDEs have decay properties, while linear hyperbolic PDEs conserve energy. Hence, it is common to choose different numerical techniques for solving and optimizing different kinds of PDEs. The type of the underlying PDE is not known a priori for a standard convolutional ResNet as it depends on the trained weights. This renders ensuring the stability of the trained network and the choice of adequate time integration methods difficult. These considerations motivate us to restrict the convolutional ResNet architecture a-priori to discretizations of nonlinear PDEs that are stable.

In our numerical examples, our new architectures lead to an adequate performance despite the constraints on the networks. In fact, using only networks of relatively modest size, we obtain results that are close to those of state-of-the-art networks. This may not hold in general, and future research will show which types of architectures are best suited for a learning task at hand. Our intuition is that, e.g., hyperbolic networks may be preferable over parabolic ones for image extrapolation tasks to ensure the preservation of edge information in the images. In contrast to that, we anticipate parabolic networks to perform superior for tasks that require filtering, e.g., image denoising. Another important direction is to quantitatively compare the architectures proposed here to existing ones with respect to aspects other than classification accuracy, e.g., robustness to adversarial attacks.

We note that our view of CNNs mirrors the developments in PDE-based image processing in the 1990s. PDE-based methods have since significantly enhanced our mathematical understanding of image processing tasks and opened the door to many popular algorithms and techniques. We hope that continuous models of CNNs will result in similar breakthroughs and, e.g., help streamline the design of network architectures and improve training outcomes with less trial and error.

Acknowledgements L.R. is supported by the U.S. National Science Foundation (NSF) through awards DMS 1522599 and DMS 1751636 and by the NVIDIA Corporation's GPU grant program. We thank Martin Burger for outlining how to show stability using monotone operator theory and Eran Treister and other contributors of the Meganet package. We also thank the Isaac Newton Institute (INI) for Mathematical Sciences for support and hospitality during the program on Generative Models, Parameter Learning and Sparsity (VMVW02) when work on this paper was undertaken. INI was supported by EPSRC Grant Number: LNAG/036, RG91310.

References

1. Ambrosio, L., Tortorelli, V.M.: Approximation of functionals depending on jumps by elliptic functionals via gamma-convergence. Commun. Pure Appl. Math. **43**(8), 999–1036 (1990)
2. Ascher, U.: Numerical Methods for Evolutionary Differential Equations. SIAM, Philadelphia (2010)
3. Ascher, U., Mattheij, R., Russell, R.: Numerical Solution of Boundary Value Problems for Ordinary Differential Equations. SIAM, Philadelphia (1995)
4. Bengio, Y., et al.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**(1), 1–127 (2009)
5. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
6. Biegler, L.T., Ghattas, O., Heinkenschloss, M., Keyes, D., van Bloemen Waanders, B.: Real-Time PDE-Constrained Optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2007)

7. Borzì, A., Schulz, V.: Computational Optimization of Systems Governed by Partial Differential Equations, vol. 8. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2012)
8. Chan, T.F., Vese, L.A.: Active contours without edges. IEEE Trans. Image Process. **10**(2), 266–277 (2001)
9. Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., Holtham, E.: Reversible architectures for arbitrarily deep residual neural networks. In: AAAI Conference on AI (2018)
10. Chaudhari, P., Oberman, A., Osher, S., Soatto, S., Carlier, G.: Deep relaxation: partial differential equations for optimizing deep neural networks. arXiv preprint arXiv:1704.04932, (2017)
11. Chen, T. Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. arXiv preprint arXiv:1806.07366 (2018)
12. Chen, Y., Pock, T.: Trainable nonlinear reaction diffusion: a flexible framework for fast and effective image restoration. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1256–1272 (2017)
13. Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., Usunier, N.: Parseval networks: improving robustness to adversarial examples. In: Proceedings of the 34th International Conference on Machine Learning, Vol. 70, pp. 854–863. JMLR. org, (2017)
14. Coates, A., Ng, A., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, pp. 215–223 (2011)
15. Combettes, P. L., Pesquet, J.-C.: Deep neural network structures solving variational inequalities. arXiv preprint arXiv:1808.07526 (2018)
16. Combettes, P. L., Pesquet, J.-C.: Lipschitz certificates for neural network structures driven by averaged activation operators. arXiv preprint arXiv:1903.01014v2 (2019)
17. Dundar, A., Jin, J., Culurciello, E.: Convolutional clustering for unsupervised learning. In: ICLR (2015)
18. Gomez, A. N., Ren, M., Urtasun, R., Grosse, R. B.: The reversible residual network: backpropagation without storing activations. In: Advances in Neural Information Processing Systems, pp. 2211–2221 (2017)
19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
20. Goodfellow, I. J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
21. Haber, E., Ruthotto, L.: Stable architectures for deep neural networks. Inverse Probl. **34**, 014004 (2017)
22. Haber, E., Ruthotto, L., Holtham, E.: Learning across scales—a multiscale method for convolution neural networks. In: AAAI Conference on AI, pp. 1–8, arXiv:1703.02009 (2017)
23. Hansen, P.C., Nagy, J.G., O'Leary, D.P.: Deblurring Images: Matrices, Spectra and Filtering. SIAM, Philadelphia (2006)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
25. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: 36th International Conference on Machine Learning, pp. 630–645 (2016)
26. Hernández-Lobato, J.M., Gelbart, M.A., Adams, R.P., Hoffman, M.W., Ghahramani, Z.: A general framework for constrained bayesian optimization using information-based search. J. Mach. Learn. Res. **17**, 2–51 (2016)
27. Herzog, R., Kunisch, K.: Algorithms for PDE-constrained optimization. GAMM-Mitteilungen **33**(2), 163–176 (2010)
28. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.-R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
29. Horn, B.K., Schunck, B.G.: Determining optical flow. Artif. Intell. **17**(1–3), 185–203 (1981)
30. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: 32nd International Conference on Machine Learning, pp. 448–456 (2015)
31. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)
32. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **61**, 1097–1105 (2012)
33. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. Handb. Brain Theory Neural Netw. **3361**, 255–258 (1995)
34. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
35. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems, pp. 253–256 (2010)
36. Li, Q., Chen, L., Tai, C., Weinan, E.: Maximum principle based algorithms for deep learning. J. Mach. Learn. Res. **18**(1), 5998–6026 (2017)
37. Modersitzki, J.: FAIR: Flexible Algorithms for Image Registration. Fundamentals of Algorithms. SIAM, Philadelphia (2009)
38. Moosavi-Dezfooli, S. M., Fawzi, A., F, O.: arXiv, and 2017. Universal adversarial perturbations. openaccess.thecvf.com
39. Mumford, D., Shah, J.: Optimal approximations by piecewise smooth functions and associated variational-problems. Commun. Pure Appl. Math. **42**(5), 577–685 (1989)
40. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: automated whitebox testing of deep learning systems. In: 26th Symposium on Oper. Sys. Princ., pp. 1–18. ACM Press, New York (2017)
41. Perona, P., Malik, J.: Scale-space and edge detection using anisotropic diffusion. IEEE Trans. Pattern Anal. Mach. Intell. **12**(7), 629–639 (1990)
42. Raina, R., Madhavan, A., Ng, A. Y.: Large-scale deep unsupervised learning using graphics processors. In: 26th Annual International Conference, pp. 873–880. ACM, New York (2009)
43. Rogers, C., Moodie, T.: Wave Phenomena: Modern Theory and Applications. Mathematics Studies. Elsevier Science, North-Holland (1984)
44. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386–408 (1958)
45. Rudin, L.I., Osher, S., Fatemi, E.: Nonlinear total variation based noise removal algorithms. Phys. D **60**(1–4), 259–268 (1992)
46. Scherzer, O., Grasmair, M., Grossauer, H., Haltmeier, M., Lenzen, F.: Variational Methods in Imaging. Springer, New York (2009)
47. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: a large data set for nonparametric object and scene recognition. IEEE Trans. Pattern Anal. Mach. Intell. **30**(11), 1958–1970 (2008)
48. Weickert, J.: Anisotropic diffusion in image processing. Stuttgart (2009)
49. Weinan, E.: A proposal on machine learning via dynamical systems. Commun. Math. Stat. **5**(1), 1–11 (2017)
50. Yang, S., Luo, P., Loy, C. C., Shum, W. K., Tang, X.: Deep visual representation learning with target coding. In: AAAI Conference on AI, pp. 3848–3854 (2015)
51. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)

Lars Ruthotto is an assistant professor of mathematics and computer science at Emory University in Atlanta, GA. He received his diploma and his Ph.D. in mathematics from the University of Münster in 2010 and 2012, respectively. His research interests include numerical analysis (particularly numerical methods for optimization, linear algebra, and partial differential equations) and scientific computing with applications in medical and geophysical imaging and machine learning.

Eldad Haber is an NSERC Industrial Research Chair at the University of British Columbia, Vancouver, Canada. Eldad is working in the field of computational inverse problems with applications in machine learning, geosciences, and medical imaging. Over the last 20 years, Eldad has written various commercial software packages that have been widely adopted by industry. Eldad has written or co-authored over 150 peer-reviewed publications on computational problems and is a U.S. Department of Energy Early Career Award recipient. After completing his Ph.D, he spent several years as a research scientist with Schlumberger and nine years at Emory University in Atlanta at the Department of Mathematics and Computer Science. In 2011, Eldad co-founded Computational Geosciences Inc. and in 2017 he co-founded Xtract.ai.